ECE 344L Laboratory 1

MIPS Assembly Language Programming

SPRING 2020

AUTHOR: DAVID KIRBY

DUE DATE: 14 FEBRUARY 2020

Laboratory 1
MIPS Software Development

Due Date: 14Feb2020 Name: David Kirby

Points: 100 Points
Work individually.

Objective: The purposes of this laboratory are to develop your MIPS programing skills and to continue to developing your proficiency in using the assembler/linker/loader system and its debugging capabilities. In this lab, you will develop assembly source that will be assembled, downloaded, and executed. The MIPS instruction set information distributed in class will be very useful for this software development process. (All class slides and reference materials are posted on UNM Learn.)

Activities: For this assignment, you will write a main routine in which at the end of execution:

v0 = the sum of even positive integers from 0 to 600. **# calculate this sum**

t0 = v0* v0 **# square the sum**

if(t0 has a greater value than can be represented in a 32 bit word)

v1 = -1

else

v1 = t0

Note: end your main routine with an endless loop, similar to what was implemented in Math.S

Repeat the exercise, but now let v0 = sum of even integers from 1 to 400 and t0 = v0 * v0, as before.

Documentation: Your lab activities must be documented following the guidelines that are provided on the course UNM Learn site. You must demonstrate also that your project functions properly to one our TAs, who will then sign your copy of this assignment sheet. We have sent out a message which notifies you the when they will be available in the lab to either help you, or witness your success. You may also contact them to arrange for another time to meet, if absolutely necessary, though you are encouraged to start work on the lab as soon as possible to avoid last minute time crunches.

Suggestion: Keep all of your files on a USB memory device as there is no guarantee that any information you store on lab machines will be preserved. On occasion, the machines must cleaned and reloaded, so any information stored on them will be lost.

TA: Sajay Krishnan

Introduction

The objective of this lab was to help develop our knowledge of MIPS assembly language and to give us practice using the software development tools. We took the MIPS instructions cheat sheet handed out in class and were tasked with implementing a series of arithmetic operations including addition and multiplication, all while testing for overflow errors.

Solution Methodology

Part I: Sum of Even Numbers 0 - 600

Following the procedures given in the assignment sheet provided on UNM Learn, we created a new project in MPLab version 8.92 using the Project Wizard. We designed an Assembly Language script based on the model we used in Lab 0, only this time we added our own code to perform the required operations (see Source Code). With the routine built, we attached and programmed the chipKit Pro MX4 board and ran the instructions. Initially, the program performed the mathematical functions and went into an infinite loop (see Figure 1). I chose to create a break point after the addition section in order to verify that the results were as to be expected before they were passed to the multiplication section.

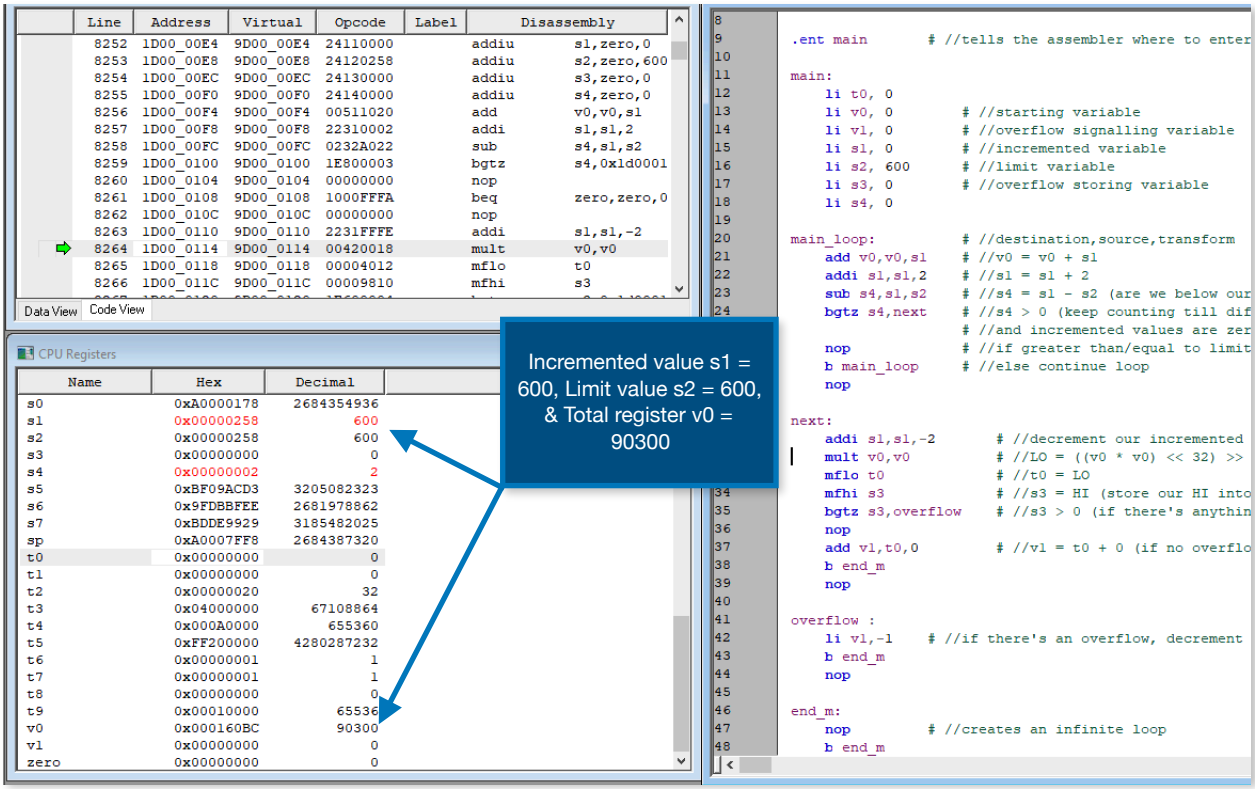

FIGURE 1

Part II: Squaring the Results from Part I

The next step seemed relatively straight forward - square the results of the previous part; however, when implementing the arithmetic, we ran into the issue of bit overflow. Our MIPS processor is 32-bits and therefore our registers only hold 32-bits; anything larger will cause an overflow. The result from Part I, 90300, squared would result in approximately 8 billion,

unfortunately, our 32-bit register can only store numbers roughly as large as 4 billion. In order to solve this, we first needed to detect the overflow from our multiplication, and then decrement our results until we reached a value that could be stored without overflow $2^{32} - 1 = 4{,}294{,}967{,}295$.

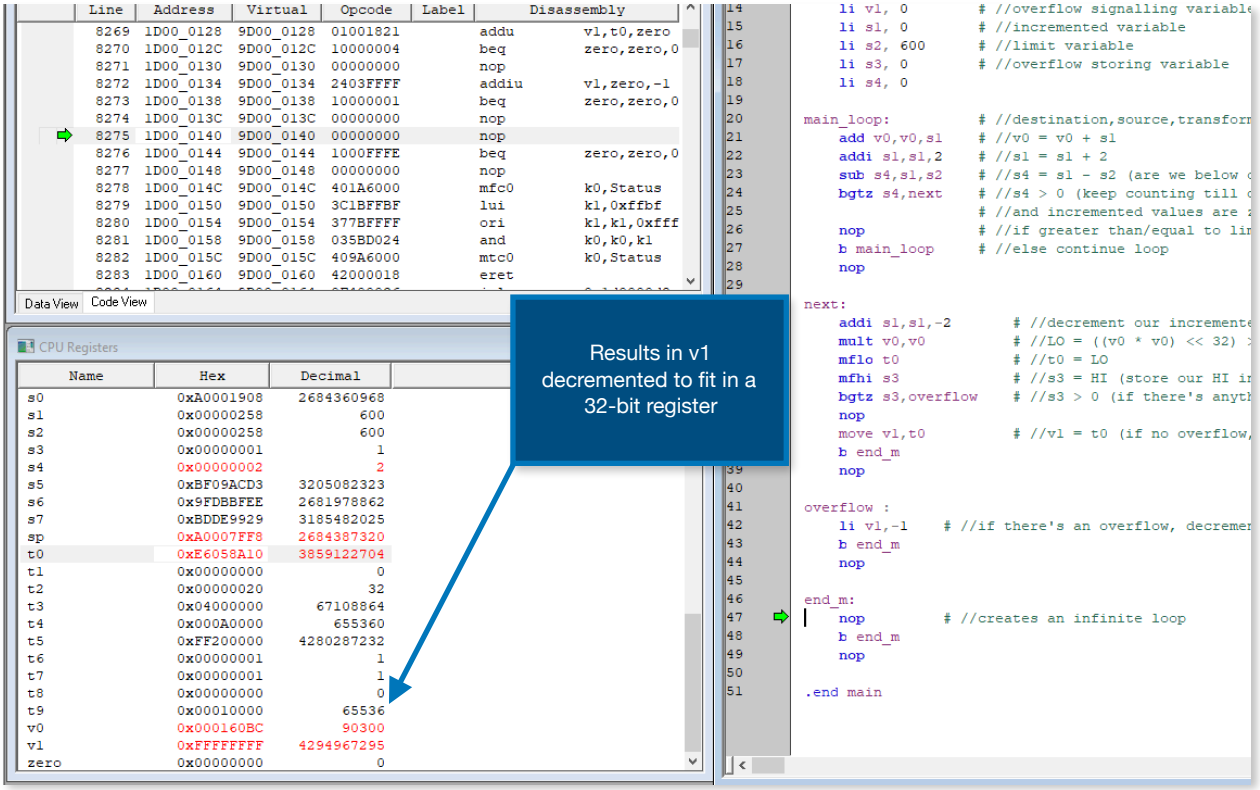

FIGURE 2

These values are consistent with the calculations as performed in MATLAB (Figure 3).

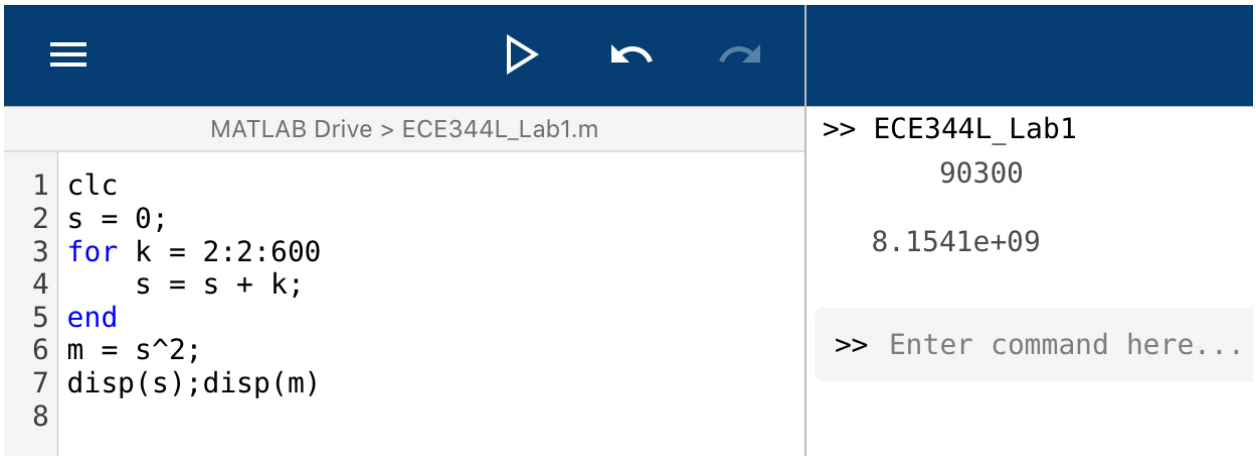

FIGURE 3

Part III: Sum of Even Numbers 1 - 400

For the final part, we repeated the processes of Parts I and II, but now implemented from 1 - 400. The reasoning for this was two-fold - we no longer had to worry about bit overflow as the result from the addition, squared is no longer too large to store in a 32-bit register, and we were presented with the challenge of implementing addition of even numbers, but starting on an odd number. Our results for this computation are shown in Figure 4.

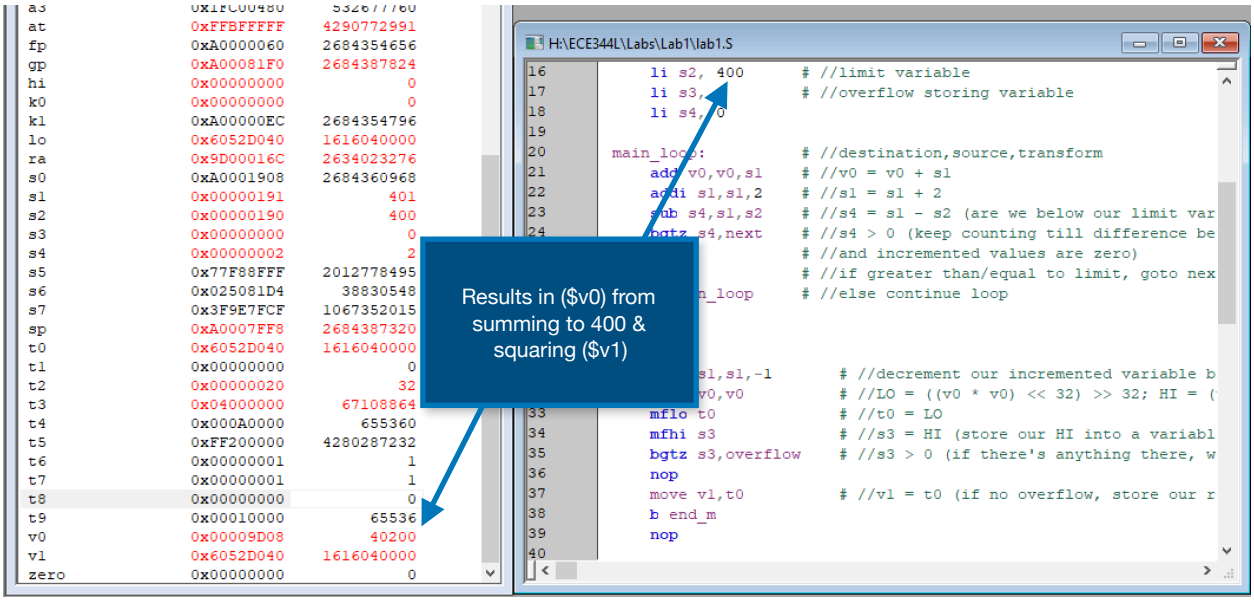

FIGURE 4

Source Code

```
/*****************************************************************************/
/*                                                                           */
/*        ECE 344L       -    Microprocessors     -         Spring 2020      */
/*                                                                           */
/*        lab1.S         MIPS Assembly Addition & Multiplication            */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*        Author:        David Kirby                                        */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*        Detailed File Description:                                         */
/*             Implements a loop of arithmetic operations and tests for      */
/*             overflow control.                                             */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*        Revision History:                                                  */
/*             Changed bgez to bgtz to correct for over-instantiation        */
/*                                                                           */
/*****************************************************************************/
```

```
#include <p32xxxx.h>

.globl main

.text                   # tells the assembler that we are in the text section
.set noreorder          # tells the assembler to keep the order in which we have put it
.ent main               # tells the assembler where to enter the routine called main

main:
     li t0, 0
     li v0, 0           # starting register
     li v1, 0           # overflow signaling register
     li s1, 0           # incremented register
     li s2, 600         # limit register
     li s3, 0           # overflow storing register
     li s4, 0

main_loop:              # destination, source, transform
     add v0,v0,s1       # v0 = v0 + s1
     addi s1,s1,2       # s1 = s1 + 2
     sub s4,s1,s2       # s4 = s1 - s2 (are we below our limit register)
     bgtz s4,next       # s4 > 0 (keep counting till difference between our limit
                        # and incremented values are zero)
     nop                # if greater than/equal to limit, goto next
     b main_loop        # else continue loop
     nop

next:
     addi s1,s1,-2          # decrement our incremented register back to our limit
     mult v0,v0             # LO = ((v0 * v0)<<32) >> 32;HI = (v0 * v0)>>32;
     mflo t0                # t0 = LO
     mfhi s3                # s3 = HI (store HI into a register to be checked)
     bgtz s3,overflow       # s3 > 0 (if there's a value, we have overflow, decrement)
     nop
     move v1,t0             # v1 = t0 (if no overflow, store our result to v1)
     b end_m
     nop

overflow :
     li v1,-1           # if there's an overflow, decrement
     b end_m
     nop

end_m:
     nop                # creates an infinite loop
     b end_m
     nop

.end main
```

Conclusion

Laboratory 1 was designed to familiarize students with the MPLab 8.92 development software and its debugger capabilities. It allowed students to take the MIPS instructions cheat sheet handed out in class and implement a series of arithmetic operations including addition and multiplication, all while testing for overflow errors. We made use of various registers and the HI and LO portions of the multiplication function to detect and store overflows.